Technical Bulletin

Part No. 74-0128

DataStage MS OLEDB

This technical bulletin describes Release 2.0 of the DataStage MS OLEDB stage, formerly called the DataStage OLE DB plug-in. This stage to read and write data to and from any type of data source. It also creates local multidimensional cubes, and loads the cubes with the data from the underlying database.

Printing History

First Edition (74-0128) for Release 1.0, July 1999
Second Edition (74-0128) for Release 1.1, March 2000
Third Edition (74-0128) for Release 2.0, July 2000
Updated for Release 2.0, August 2002
Updated for Release 2.0, August 2003

How to Order Technical Documents

To order copies of documents, contact your local Ascential subsidiary or distributor, or call our main office at (508) 366-3888.

Documentation Team: Marie E. Hedin

Introduction

This technical bulletin describes the following for Release 2.0 of DataStage MS OLEDB for DataStage Release 7.0:

- Functionality
- Installation
- Defining the MS OLEDB stage
- Connecting to an MS OLEDB data source
- Creating and populating cubes
- Defining character set mapping
- Defining input links
- Creating and populating cubes
- Defining output links
- MS OLEDB server data type support
- Stored procedure support
- CREATE CUBE and INSERT INTO statements

MS OLEDB provides a user-friendly client GUI. You can install the GUI separately from MS OLEDB on the server. Or, you can use the server MS OLEDB stage without installing the custom GUI on the client. If it is not installed, you can still use the property grid interface.

MS OLEDB is a passive stage that lets DataStage retrieve information from any type of information repository, such as a relational source, an ISAM file, a personal database, or a spreadsheet. This stage uses OLE DB SDK that is part of Microsoft Data Access SDK (MDAC). For more information about Microsoft Data Access, consult your MDAC documentation.

Note: You can also use this stage to create a local cube file, and load the cube with the data from the underlying database. You can use the CREATE CUBE and INSERT INTO statements to do this (see the examples on page 35).

However, you cannot create a cube on the OLAP server yet since MSOLAP does not currently support this. Additionally, there is no GUI browser for local cube files, but you can modify the MDX sample application that comes with the Microsoft SQL Server for OLAP Services software to browse local cube files.

You can use any number of input, output, or reference output links with this stage:

- Input links specify the data you are writing, which is a stream of rows to be loaded into the data source.

- Output links specify the data you are extracting, which is a stream of rows to be read from the data source. You can specify the data on an input or an output link using an SQL statement constructed by DataStage or a user-defined query.
- Each reference output link represents rows that are key read from the data source. The link uses the key columns in a WHERE clause of the SELECT statement that is constructed by DataStage or specified by the user.

The MS OLEDB stage can read and write data directly without SQL statements. Or, the stage can use SQL statements generated by the stage or user-defined SQL statements.

For More Information. For information about using stages and related topics, see the following table:

If you want information on…	Then see…
Using a stage in a DataStage job	*DataStage Server Job Developer's Guide*
Pivot Table Service	Microsoft SQL Server for OLAP Services or Microsoft Data Access Component SDK documentation
Cubes	"Pivot Table Service" in the online Microsoft SQL Server for OLAP Services documentation
Using synchronous initialization and isolation levels	OLE DB documentation
Using MetaStage	*MetaStage User's Guide*
Using NLS	*DataStage Designer Guide* or *DataStage NLS Guide*

Functionality

The MS OLEDB stage can do the following:

- Create local multidimensional databases called local cube files, and load these cubes with the data from the underlying database.
- Support stream input, stream output, and reference output links.
- Specify the number of rows to retrieve from the data source.
- Specify the number of rows to update at one time.

- Specify the isolation level used for transactions.
- Specify the number of rows to write before committing them.
- Control the type of tracing information to add to the log.
- Specify which generated or user-defined SQL statements to execute for reading or writing data.
- Specify additional clauses to append to the generated SQL statements.
- Specify SQL statements to run before and after processing job data rows.
- Specify which mode to use to retrieve or insert data.
- Browse source or target data using the GUI.
- Support MetaStage. For information, see *MetaStage User's Guide.*
- Import table and column definitions from the target OLE DB data source and store them in the DataStage Repository. For more information about meta data import, see *DataStage Server Job Developer's Guide*.
- Use NLS (National Language Support). For information, see *DataStage NLS Guide*.

Terminology

The following table lists the Microsoft OLAP Services Bulk Load terms used in this document:

Term	Description
Dimension	A collection of measures and a set of dimensions. Each measure has an aggregate function, and each dimension contains one or more level. Optionally, dimensions can include multiple hierarchies. Each hierarchy contains levels.
Objects	Distinct items in the database such as dimensions, variables, formulas, and so forth. Used to create OLAP applications to access, manipulate, and display data stored in a multidimensional database management system.
OLAP	Online Analytical Processing. This processing uses multidimensional data.
UNC	Universal Naming Convention. A PC format that specifies the location of resources on a network.

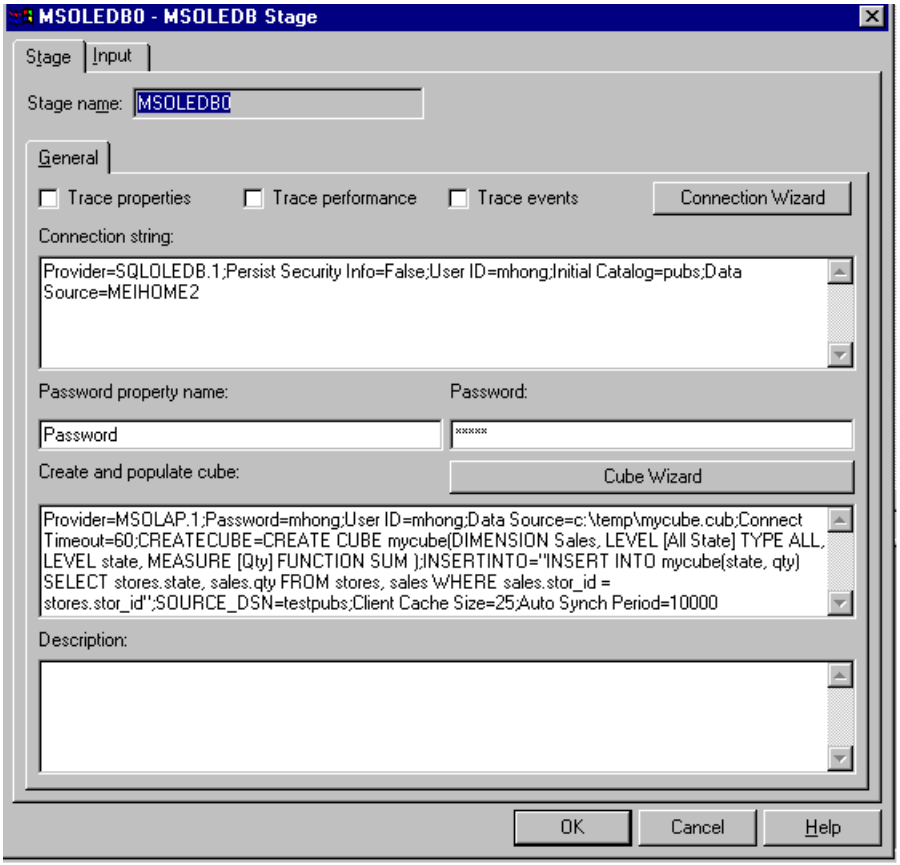
Term	Description
Valueset	An object that contains a list of dimension values for a particular dimension.
Variable	An object that stores the actual data. All of the data in a variable represents the same unit of measurement with the same data type. Typically a variable is a multidimensional array, from which you can uniquely select any data value within it by specifying one member from each of its dimensions.

Installing the Plug-In

For instructions and information supporting the installation, see *DataStage Plug-In Installation and Configuration Guide*.

Defining the MS OLEDB Stage

When you use the custom GUI to edit an MS OLEDB stage, the **MSOLEDB Stage** dialog box appears:



This dialog box has the **Stage**, **Input**, and **Output** pages (depending on whether there are inputs to and outputs from the stage):

- **Stage.** This page displays the name of the stage you are editing. The **General** page defines the MS OLEDB tracing, provider, connection, and login information. The properties on this page define the connection to the OLE DB data source. Additionally, you can specify information to create and populate a cube. For connection details, see "Connecting to an OLE DB Data Source" on page 6.

The **NLS** page defines a character set map to use with the stage. This page appears only if you have installed NLS for DataStage. For details, see "Defining Character Set Mapping" on page 14.

Note: You cannot change the name of the stage from this dialog box. For details on changing stage names, see *DataStage Designer Guide*.

- **Input.** This page is displayed only if you have an input link to this stage. It specifies the data source to use and the associated column definitions for each data input link. It also specifies how data is written, the transaction isolation level, array size, and tracing information used to write data to a data source.
- **Output.** This page is displayed only if you have an output or reference link to this stage. It specifies the data sources to use and the associated column definitions for each data output link. It also specifies how to read the data, the transaction isolation level, array size, and tracing information used to read the data.

The main phases in defining an MS OLEDB stage from the **MSOLEDB Stage** dialog box are as follows:

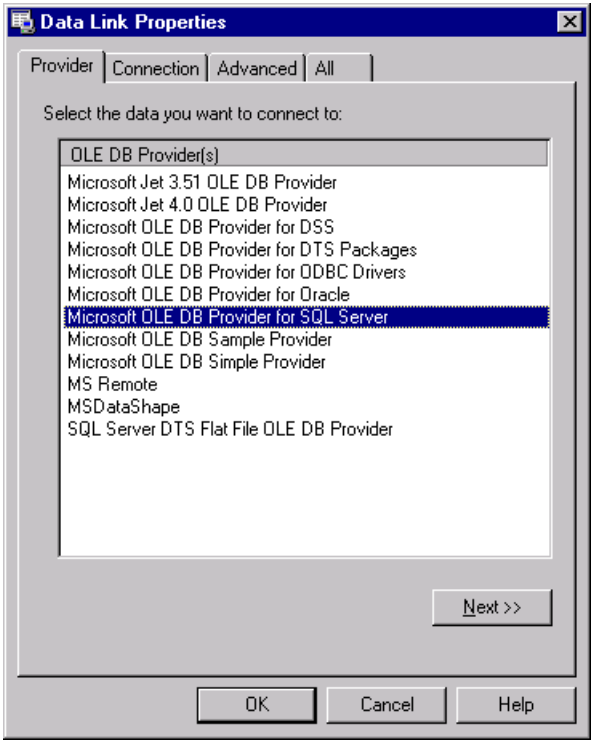
1. Connect to an OLE DB data source. (See page 6).
2. Optional. Generate a new connection string. (See page 8).
3. Optional. Use the cube wizard to define a cube. (See page 8).
4. Optional. Define a character set map (see page 14).
5. Define the data on the input links (see page 15).
6. Define the data on the output links (see page 22).

Connecting to an OLE DB Data Source

The OLE DB connection parameters are set on the **General** page of the **Stage** page. Specify the appropriate information using the following button and fields:

- **Trace properties.** If selected, adds tracing information about stage and link properties to the log at run time.
- **Trace performance.** If selected, adds tracing information about performance, such as link timing information, to the log.
- **Trace events.** If selected, adds tracing information about important events, such as initialization messages, to the log.

- **Connection Wizard.** Opens the **Data Link Properties** dialog box, where you specify the data you want to connect to. This standard Microsoft OLE DB connection dialog box contains the **Provider**, **Connection**, **Advanced**, and **All** pages. Click **Next >>** to specify additional properties, such as the machine name, for the selected provider.



- **Connection string.** Displays the connection information that you entered using the **Connection Wizard**. If you do not use the wizard, you can enter the information directly into this field.
- **Password property name.** The property name for the provider password, if the provider requires it. If you do not use the wizard, you can also enter the information directly into the **Connection string** field, but it will not be encrypted unless you enter it in the **Password property name** field.
- **Password.** The encrypted password for the provider. For security, it displays asterisks instead of the value you enter. If you do not use the wizard, you can also enter the information directly into the **Connection**

string field, but it will not be encrypted unless you enter it in the **Password** field.

- **Cube Wizard.** Click to open the **Data Link properties** dialog box, where you connect to the Microsoft OLE DB provider for OLAP Services. This is the same dialog box that you open from the **Connection Wizard** for the **General** page of the **Stage** page with the additional provider. You then go to the **All** page to specify the parameters to create and populate a local cube file. For details, see "Creating and Populating Cubes" on page 8.
- **Create and populate cube.** Displays the connection information that you entered using the **Cube Wizard**. If you do not use the wizard, you can enter the information directly into this field.
- **Description.** Optionally, describe the purpose of the MS OLEDB stage.

Creating and Populating Cubes

The MS OLEDB stage loads data to and from a database table. It also provides functionality to create local multidimensional databases called local cube files, and loads these cubes with the data from the underlying database.

It uses the Microsoft Pivot Table Service Provider (MSOLAP) to create and populate the cubes. MSOLAP does not distinguish between the creation of a cube and inserting into or populating the cube. This is because an INSERT INTO statement must be used with the CREATE CUBE statement to provide structure for a multidimensional database.

Note: Currently MSOLAP does not support cube creation on the OLAP server. Therefore, a cube is a local cube file.

You can view a cube file as a multidimensional data repository. Each cube file ends in a .cub extension and can contain multiple cubes. Each cube in the file can contain multiple catalogs.

For example, suppose the Sales.Cub is the multidimensional data repository. The CREATE CUBE statement creates the Sales_USA, Sales_India, Sales_UK, and Sales_Rest cubes. The INSERT INTO statement creates the North and South catalogs inside Sales_USA, the All catalog inside Sales_India, and the All catalog inside Sales_Rest. Every INSERT INTO statement must be used with a corresponding

CREATE CUBE statement. If the specified cube already exists, the statement is ignored. Otherwise, it is created and a new catalog specified by the INSERT INTO statement is created in that cube.

To edit the data link properties to create the cube:

1. Click **Cube Wizard** from the **General** page of the **Stage** page to connect to the provider. You can also enter the connection string information in the **Create and populate cube** field on the same page. In this case, the cube is created at the end of the table loading process.

   Note: Release 2.0 of the MS OLEDB stage must have only one input link and no output links to create the cube successfully. You should not use multiple input links or input and output links in the same MS OLEDB stage.

2. Select **Microsoft OLE DB Provider for OLAP Services**. The **Data Link Properties** dialog box appears.
3. Click the **All** tab, then specify the parameters to create and populate a local cube file.
4. Select the parameter, then click **Edit Value…** to enter the appropriate information for the data link initialization properties described in the following table. The following table describes the most important properties. Supply information for the Data Source, CREATECUBE, INSERTINTO, SOURCE_DSN, User ID, and Password properties:

Data Link Properties

Property	Description
ARTIFICIALDATA	The artificial aggregate values calculated instead of calculating real values. These values are calculated using a simple algorithm when the first character of this string is Y, T, or a numeric digit other than 0.
Data Source	The name of the local cube file you want to create. The default extension for a local cube file is .cub, but any extension can be used.
Initial Catalog	The name of the default initial database catalog. Use this property unless you are creating a local cube. The value is used when a session is established, but you cannot change the value during the session.

Data Link Properties (Continued)

Property	Description
CREATECUBE	The CREATE CUBE statement to create a local cube file. You must use this property if you also use the INSERTINTO and SOURCE_DSN properties. These three properties are always used together. This value is used when a session is established, but you cannot change it during the session.
INSERTINTO	The INSERT INTO statement to populate a local cube file that was created using the CREATE CUBE statement.
SOURCE_DSN	The ODBC connection string, OLE DB connection string, or DSN for the source relational database, used only when creating a local cube file.
User ID	The ODBC UID for the source database, used only when creating a local cube file.
Password	The ODBC PWD for the source database, used only when creating a local cube file.
Auto Synch Period	Specifies how often queries are made to the source database. The default value on the server is 10,000 milliseconds (10 seconds). By setting this property to a null value or 0, automatic synchronization is turned off, and synchronization does not occur at a constant interval. The frequency of synchronization depends on client activity. Some client queries are resolved solely from the client cache. Therefore, a high value can cause more frequent query results that do not reflect recent updates in the data source. A low value can reduce the likelihood of these events. However, a low value can impede performance. The lowest valid, nonzero value is 250 milliseconds. A value of 250 milliseconds is used for any value from 1 to 249. This value is used when a session is established, but you cannot change it during the session.
Cache Policy	Specifies information about memory.

Data Link Properties (Continued)

Property	Description
Client Cache Size	The specified number of kilobytes (KB) of memory in the client cache. If set to 0, the client cache can use unlimited memory. If set to a value from 1 to 99, the client cache can use the specified percentage of total available virtual memory (physical and page file). If this property is set to 100 or more, the client cache can use up to the specified KB of memory. This value is used when a session is established, but you cannot change it during the session.
CompareCaseSensitive StringFlags	The flags used in case-sensitive string comparisons to control string comparisons and sort order.
CompareCaseNotSensitive StringFlags	The flags used in string comparisons that are not case-sensitive to control string comparisons and sort order. This property is used more frequently in NLS versions. The default is the value of the CompareCaseSensitive StringFlags registry on the client if this registry exists.
Default Isolation Mode	If the first character of this string is Y, T, or a numeric digit other than 0, the isolation level is isolated. Otherwise, the isolation level is determined by the cursor type requested by the rowset properties. For more information about isolation levels, see the OLE DB documentation.
Execution Location	The location of the query execution. Use one of the following values: 0 - The default value. This is equivalent to a value of 1. 1 - The automatic selection of query execution location, either client or server. 2 - The query executes on the client. 3 - The query executes on the server. Exceptions include queries that contain session-scoped calculated members, user-defined sets, or user-defined functions.

Data Link Properties (Continued)

Property	Description
Large Level Threshold	Specifies whether dimension levels are sent from the server to the client incrementally or in their entirety. Dimension levels that contain a number of members greater than or equal to the value of this property are sent incrementally. A level that contains fewer members than the value of this property is sent to the client in its entirety. This helps manage client memory usage. The default value is set on the server in the **Large level defined as** box in the **Data Link Properties** dialog box. The minimum value is 10. Settings less than 10 are ignored, and the minimum value is used. In this case, no error is returned.
Locale Identifier	The ID for the locale (LCID), which the client can modify by setting the DBPROP_INIT_LCID property. Pivot Table Service can have only one LCID per Windows process. The LCID must be installed in Control Panel in Windows, or the attempt to set the LCID fails. By default, the DBPROP_INIT_LCID is reported as null.
SOURCE_DSN_SUFFIX	The suffix used only when creating or connecting to a local cube. This value is not stored in the local cube file. This property is useful for separating data persisted in the local cube file from data used only for the session. (Session data includes user account and password.)
USEEXISTINGFILE	Specifies whether to connect to the existing local cube. If the first character of this value is Y, T, or a numeric digit other than 0, and the cube file specified in the Data Source property already exists, the CREATECUBE and INSERTINTO properties are ignored. A connection to the existing local cube is established. If this value is not used, or the first character of this value is not Y, T, or a numeric digit other than 0, and the cube file specified in the Data Source property already exists, the statements in the CREATECUBE and INSERTINTO properties overwrite the existing cube file.

Data Link Properties (Continued)

Property	Description
Writeback Timeout	The number of seconds before an update occurs. The attempt to communicate updates is triggered by a commit, which begins a counting of seconds. The counting continues until the commit is successful or the specified number of seconds is reached. If this value is reached, the commit fails and the update does not occur. You can then attempt another commit or a rollback.

For details about the CREATE CUBE and INSERT INTO statements, see page 30 and page 32. For more information about Pivot Table Service, see the documentation for Microsoft SQL Server for OLAP Services and Microsoft Data Access Component SDK.

Defining Character Set Mapping

You can define a character set map for a stage. Do this from the **NLS** tab that appears on the **Stage** page. The **NLS** page appears only if you have installed NLS.

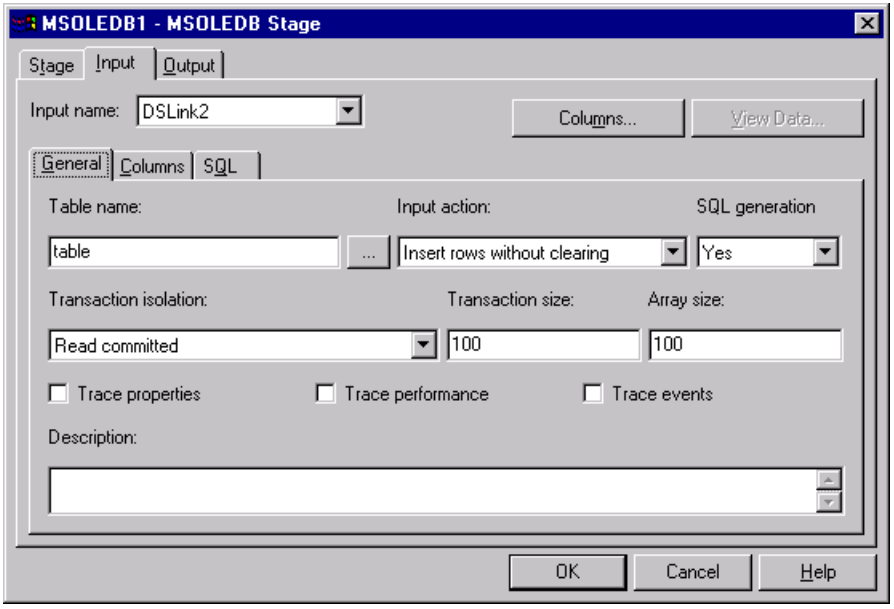
Specify information using the following button and fields:

- **Map name to use with stage.** The default character set map is defined for the project or the job. You can change the map by selecting a map name from the list.
- **Use Job Parameter… .** Specifies parameter values for the job. Use the format #*Param*#, where *Param* is the name of the job parameter. The string #*Param*# is replaced by the job parameter when the job is run.
- **Show all maps.** Lists all the maps that are shipped with DataStage.
- **Loaded maps only.** Lists only the maps that are currently loaded.

For more information about NLS or job parameters, see *DataStage NLS Guide* or *DataStage Designer Guide.*

Defining an Input Link

When you write data to a data source, the MS OLEDB stage has an input link. Define the properties of this link and the column definitions of the data on the **Input** page in the **MSOLEDB Stage** dialog box.



About the Input Page

The **Input** page has an **Input name** field, the **General**, **Columns**, and **SQL** pages, and the **Columns…** and **View Data…** buttons. (The **View Data…** button is disabled in this release.)

- **Input name.** The name of the input link. Choose the link you want to edit from the **Input name** drop-down list box. This list displays all the input links to the MS OLEDB stage.
- Click the **Columns…** button to display a brief list of the columns designated on the input link. As you enter detailed meta data in the **Columns** page, you can leave this list displayed.
- Click the **View Data…** button to start the Data Browser. This lets you look at the data associated with the input link. For a description of the Data

Browser, see *DataStage Designer Guide*. (The **View Data…** button is disabled in this release.)

General Page

This page is displayed by default. Enter the appropriate information for the following fields:

- **Table name.** The name of the target data source to insert data into. It is used to generate an SQL statement. It is also used when the **Input action** field is set to **Direct write**. There is no default. You can also click the **…** button at the right of the **Table name** field to browse the Repository to select the data source.
- **Input action.** Specifies which stage-generated SQL statements are used to update the target data source. Some actions require key columns to update or delete rows. Choose one of the following options:

  Direct write. Opens the data source for writing without using SQL statements. (Not all providers support this option.)

  Clear table then insert rows. Uses the SQL DELETE statement to clear the table before using the SQL INSERT statement to write the rows to the table.

  Truncate table then insert rows. Uses the SQL TRUNCATE statement to truncate the table before inserting rows.

  Insert rows without clearing. Like **Direct write**, but uses the SQL INSERT statement to write to the data source. This is the default.

  Delete existing rows only. Uses the SQL DELETE WHERE statement to delete rows from the target data source. The values of the DataStage key columns determine which rows to delete.

  Replace existing rows completely. Uses the SQL DELETE WHERE statement to delete rows from the target data source before inserting new rows.

  Update existing rows only. Uses the SQL UPDATE WHERE statement to update existing rows in the target data source.

  Update existing rows or insert new rows. Uses the SQL UPDATE WHERE statement to update existing rows. If it fails, it uses the SQL INSERT statement to insert missing rows.

Insert new rows or update existing rows. Uses the SQL INSERT statement to insert rows. If a row exists, it tries to use the SQL UPDATE WHERE statement to update the row.

- **SQL generation.** Determines if the stage generates SQL statements or uses user-defined SQL statements. Choose one of the following options:

  Yes. Specifies that the stage generates SQL statements. This is the default.

  No. Specifies that the stage uses user-defined SQL statements.

- **Transaction isolation.** Specifies the isolation level that provides the necessary consistency and concurrency control between transactions in the job and other transactions for optimal performance. Choose one of the following options:

  Read committed. Takes exclusive locks on modified data and sharable locks on all other data. Each query executed by a transaction sees only data that was committed before the query (not the transaction) began. This is the default.

  Read uncommitted. Takes exclusive locks on modified data. Transactions are not isolated from each other. So that they do not adversely affect other transactions, transactions running at this level are usually read-only.

  Repeatable read. The transaction waits until rows write-locked by other transactions are unlocked. This prevents it from reading any dirty data. The transaction holds read locks on all rows it returns to the application and write locks on all rows it inserts, updates, or deletes.

  Serializable. Takes exclusive locks on modified data and sharable locks on all other data. The transaction waits until rows write-locked by other transactions are unlocked. This prevents it from reading any dirty data.

- **Transaction size.** The number of rows that the stage processes before committing a transaction to the data source. The default is 100, meaning 100 rows are written before being committed. This field is ignored for nonlogging data sources.

- **Array size.** The maximum number of rows read at a time, that is, the array binding size. This property provides delayed update capability, meaning that changes for every row inserted are held in the cache for the rowset before the rows are updated in the data source. Delayed retrieval helps to reduce network traffic, resulting in better performance. If the value for this property exceeds the provider-specified limit, it is set to the provider limit. The default is 100.

- **Trace properties.** If selected, adds tracing information about stage and link properties to the log at run time.
- **Trace performance.** If selected, adds tracing information about performance, such as link timing information, to the log.
- **Trace events.** If selected, adds tracing information about important events, such as initialization messages, to the log.
- **Description.** Optionally enter text to describe the purpose of the link.

Columns Page

This page contains the column definitions for the data written to the data source. The **Columns** page behaves the same way as the **Columns** page in the ODBC stage. For a description of how to enter and edit column definitions, see *DataStage Designer Guide*.

SQL Page

This page displays the stage-generated or user-defined SQL statements used to write data to a data source. It contains the **Generated**, **User-defined**, **Before**, and **After** pages, which are the same as those for the **Output** page under the **SQL** page.

- **Generated.** It contains the SQL statements constructed by DataStage that are used to write data to the data source. You cannot edit these statements, but you can use **Copy** to copy them to the Clipboard for use elsewhere.
- **User-defined.** This page is displayed by default. It contains the SQL statements executed to write data to the data source**.** The GUI displays the stage-generated SQL statement on this page as a starting point. However, you can enter any valid, appropriate SQL statement. The box size changes proportionately when you resize the main window to display long SQL statements.
- **Before.** This page contains the SQL statements executed before the stage processes any job data rows. Use a semicolon (;) to separate multiple BeforeSQL statements. The SQL statement is entered in a resizable edit box. Execution occurs immediately after a successful data source connection. The **Before** and **After** pages look alike.
- **After.** This page contains the SQL statements executed after the stage processes any job data rows. Use a semicolon (;) to separate multiple AfterSQL statements. The SQL statement is entered in a resizable edit box. Execution occurs immediately after the last row is processed, before the

data source connection is terminated. The **Before** and **After** pages look alike.

Writing Data to OLE DB

The following sections describe the differences when you use generated or user-defined SQL INSERT, DELETE, or UPDATE statements to write data from DataStage to a data source. You can also execute BeforeSQL or AfterSQL statements before or after the stage processes job data rows.

Using Generated SQL Statements

By default, DataStage writes data to a data source using an SQL INSERT, DELETE, or UPDATE statement that it constructs. The generated SQL statement is automatically constructed using the DataStage table and column definitions that you specify in the input properties for this stage. The **Generated** page on the **SQL** page displays the SQL statement used to write the data.

To use a generated statement:

1. Enter a table name in the **Table name** field on the **General** page of the **Input** page.
2. Specify how you want the data to be written by choosing an option from the **Input action** list box. See "General Page" on page 16 for a description of the input actions.
3. Optional. Enter a description of the input link in the **Description** field.
4. Click the **Columns** tab on the **Input** page.
5. Edit the Columns grid to specify the column definitions for the columns you want to write. The SQL statement is automatically constructed using your chosen input action and the columns you have specified. You can now optionally view this SQL statement.
6. Click the **SQL** tab on the **Input** page, then the **Generated** tab to view this SQL statement. You cannot edit the statement here, but you can always access this tab to select and copy parts of the generated statement to paste into the user-defined SQL statement.
7. Click **OK** to close this dialog box. Changes are saved when you save your job design.

Using User-Defined SQL Statements

Instead of writing data using an SQL statement constructed by DataStage, you can enter your own SQL INSERT, DELETE, or UPDATE statement or call stored procedures for each MS OLEDB input link. Ensure that the SQL statement contains the table name, the type of input action you want to perform, and the columns you want to write.

To use your own SQL statement:

1. Set **SQL generation** to **No** on the **General** page of the **Input** page.
2. Specify how you want the data to be written by choosing an option from the **Input action** drop-down list box. See "General Page" on page 16 for a description of the input actions.
3. Click the **SQL** tab, then the **User-defined** tab. By default you see the stage-generated SQL statement. You can edit this statement or enter your own SQL statement to write data to the target data sources. This statement must contain the table name, the type of input action you want to perform, and the columns you want to write.

   When writing data, the INSERT statements must contain a VALUES clause with a parameter marker (?) for each stage input column. UPDATE statements must contain a SET clause with parameter markers for each stage input column. UPDATE and DELETE statements must contain a WHERE clause with parameter markers for the primary key columns.

   The type of SQL statement used depends on the number of parameters and key columns required. The parameter markers must be in the same order as the associated columns listed in the stage properties. Use statements such as the following:

If a statement has…	Use a statement like…
As many parameters as there are key columns	DELETE from TABLE WHERE Key1=? and Key2=?
As many parameters as there are columns	INSERT into TABLE (Col1, Col2) VALUES (?, ?)
As many parameters as there are columns and key columns	UPDATE TABLE SET Col1=?, Col2=?, Key1=?, Key2=? WHERE Key1=? and Key2=?

   The size of this box changes proportionately when the main window is resized in order to conveniently display very long or complex SQL statements.

Unless you specify a user-defined SQL statement, the stage automatically generates an SQL statement.

If you specify multiple SQL statements, each is executed as a separate transaction. End SQL statements using a semicolon (;) as the end-of-batch signal. You cannot combine multiple INSERT, UPDATE, and DELETE statements in one batch. You must execute each in a separate command batch.

4. Click **OK** to close the this dialog box. Changes are saved when you save your job design.

Using BeforeSQL Statements

You can execute SQL statements before the stage processes any job data rows. To specify SQL statements before processing any data:

1. Enter the SQL statements you want to be executed before data is processed in the text entry area on the **Before** page of the **SQL** page.

   Execution occurs immediately after a successful data source connection. If you specify multiple SQL statements, they are executed as one or more Transact-SQL command batches using a semicolon (;) as the end-of-batch signal.

2. Select the **Treat errors as non-fatal** check box to log BeforeSQL execution errors as warnings. Processing continues with the next command batch, if any. Each successful execution is committed as a separate transaction.

   If this check box is cleared, BeforeSQL execution errors are considered fatal to the job and result in a transaction rollback. The transaction is committed only if all BeforeSQL statements successfully execute.

Using AfterSQL Statements

You can execute SQL statements after the stage processes all job data rows. To specify SQL statements after processing data:

1. Enter the SQL statements you want to be executed after the data is processed in the text entry area on the **After** page of the **SQL** page.

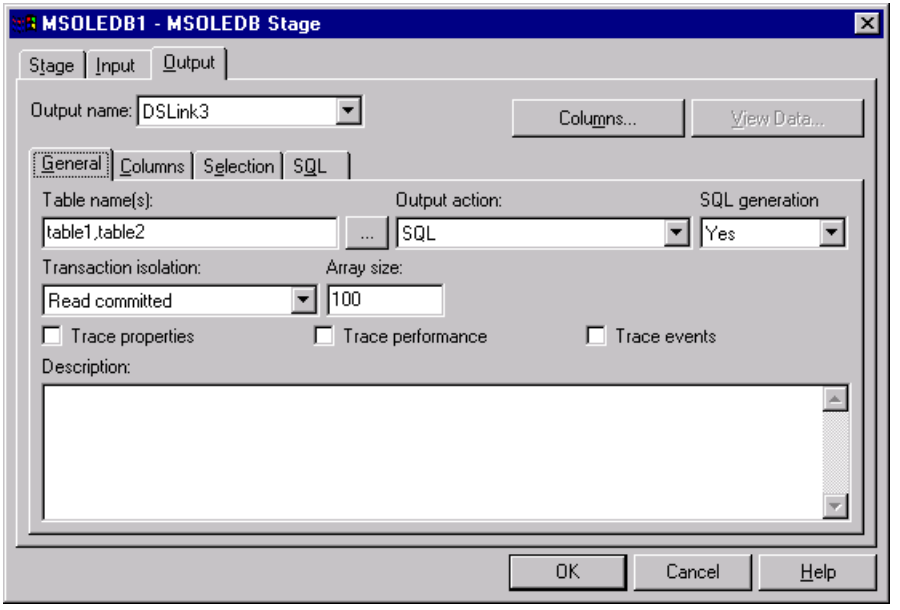
   Execution occurs immediately before the data source connection is terminated. If you specify multiple SQL statements, they are executed as one or more Transact-SQL command batches using a semicolon (;) as the end-of-batch signal.

2. Select the **Treat errors as non-fatal** check box to log AfterSQL execution errors as warnings. Processing continues with the next command batch, if any. Each successful execution is committed as a separate transaction.

If this check box is cleared, AfterSQL execution errors are considered fatal to the job and result in a transaction rollback. The transaction is committed only if all AfterSQL statements successfully execute.

Defining an Output Link

When you read data from a data source, the MS OLEDB stage has an output link. Define the properties of this link and the column definitions of the data on the **Output** page in the **MSOLEDB Stage** dialog box.



About the Output Page

The **Output** page has an **Output name** field, the **General**, **Columns**, **Selection**, and **SQL** pages, and the **Columns…** and **View Data…** buttons. (The **View Data…** button is disabled in this release.) The pages displayed depend on how you specify the SQL statement to output the data.

- **Output name.** The name of the output link. Choose the link you want to edit from the **Output name** drop-down list box. This list displays all the output links from the MS OLEDB stage.

- Click the **Columns…** button to display a brief list of the columns designated on the output link. As you enter detailed meta data in the **Columns** page, you can leave this list displayed.
- Click the **View Data…** button to start the Data Browser. This lets you look at the data associated with the output link. For a description of the Data Browser, see *DataStage Server Job Developer's Guide*. (The **View Data…** button is disabled in this release.)

General Page

This page is displayed by default. Enter the appropriate information for the following fields:

- **Table names.** This field contains the names of the data sources to retrieve data from. These tables must exist or be created and populated by the BeforeSQL statements. You can also click the **…** button at the right of the **Table names** field to browse the Repository to select tables.

  Separate multiple table names by a comma (,). You must have select privileges on each table. There is no default.

  This property is used to generate an SQL statement. It is also used when the **Output action** field is set to **Direct read**. If **SQL generation** is set to **No**, **Table names** is ignored. You must specify **Table names** if **SQL generation** is set to **Yes**.

  Additionally, you can use a job parameter to specify the data source. For details on how to use define and use job parameters, see *DataStage Server Job Developer's Guide*.

- **Output action.** Specifies whether to retrieve data by directly reading the data source or to use SQL statements. You can only use a single SELECT statement. Choose one of the following options:

  SQL. Specifies that the data is extracted using SQL statements. This is the default.

  Direct read. Specifies that the data is extracted by directly reading the data source.

- **SQL generation.** Determines if the stage generates SQL statements or uses user-defined SQL statements to retrieve data. Choose one of the following options:

  Yes. Specifies that the stage generates an uneditable SQL statement. When this option is selected, the **Generated** page appears. You cannot

edit this statement, but you can specify the tables and columns to be output. This is the default.

No. Specifies that the stage uses user-defined SQL statements. When this option is selected, the **User-defined** page appears allowing you to edit SQL statements.

- **Transaction isolation.** Specifies the isolation level that provides the necessary consistency and concurrency control between transactions in the job and other transactions for optimal performance. Choose one of the following options:

  Read committed. Takes exclusive locks on modified data and sharable locks on all other data. Each query executed by a transaction sees only data that was committed before the query (not the transaction) began. This is the default.

  Read uncommitted. Takes exclusive locks on modified data. Transactions are not isolated from each other. So that they do not adversely affect other transactions, transactions running at this level are usually read-only.

  Repeatable read. The transaction waits until rows write-locked by other transactions are unlocked. This prevents it from reading any dirty data. The transaction holds read locks on all rows it returns to the application and write locks on all rows it inserts, updates, or deletes.

  Serializable. Takes exclusive locks on modified data and sharable locks on all other data. The transaction waits until rows write-locked by other transactions are unlocked. This prevents it from reading any dirty data.

- **Array size.** The maximum number of rows read at a time, that is, the array binding size. Delayed retrieval helps to reduce network traffic, resulting in better performance. If the value for this property exceeds the provider-specified limit, it is set to the provider limit. The default is 100.
- **Trace properties.** If selected, adds tracing information about stage and link properties to the log at run time.
- **Trace performance.** If selected, adds tracing information about performance, such as link timing information, to the log.
- **Trace events.** If selected, adds tracing information about important events, such as initialization messages, to the log.
- **Description.** Optionally enter text to describe the purpose of the output link.

Columns Page

This page contains the column definitions for the data being output on the chosen link. The column definitions are used in the order they appear in the Columns grid. The **Columns** page behaves the same way as the **Columns** page in the ODBC stage. For a description of how to enter and edit column definitions, see *DataStage Designer's Guide*.

The column definitions for output and reference links contain a key field. Key fields are used to join primary and reference inputs to a Transformer stage. MS OLEDB key reads the data by using a WHERE clause in the SQL SELECT statement.

Selection Page

This page is used primarily with generated SQL queries. It contains optional SQL SELECT clauses, such as WHERE, HAVING, or ORDER BY for the conditional extraction of data.

If you want to use the additional SQL SELECT clauses, you must enter them on the **Selection** page of the **Output** page. These clauses are appended to the SQL statement that is generated by the stage. If this link is a reference link, only the WHERE clause is enabled.

The **Selection** page is divided into two areas (panes). You can resize an area by dragging the split bar.

- **WHERE clause.** This text box allows you to insert an SQL WHERE clause to specify criteria that the data must meet before being selected.
- **Other clauses.** This text box allows you to insert a HAVING or an ORDER BY clause.

SQL Page

This page displays the stage-generated or user-defined SQL statements used to read data from OLE DB. It contains the **Generated**, **User-defined**, **Before**, and **After** pages, which are the same as those for the **Input** page under the **SQL** page.

- **Generated.** This contains the SQL statements constructed by DataStage as a result of the **Output action** from the **General** page of the **Output** page. You cannot edit these statements, but you can use **Copy** to copy them to the Clipboard for use elsewhere.
- **User-defined.** This page is displayed by default. It contains the SQL statements executed to read data from the data source. The GUI displays the stage-generated SQL statement on this page as a starting point. However,

you can enter any valid, appropriate SQL statement. The box size changes proportionately when you resize the main window to display long SQL statements.

- **Before.** This page contains the SQL statements executed before the stage processes any job data rows. Use a semicolon (;) to separate multiple BeforeSQL statements. The SQL statement is entered in a resizable edit box. Execution occurs immediately after a successful data source connection. The **Before** and **After** pages look alike.
- **After.** This page contains the SQL statements executed after the stage processes any job data rows. Use a semicolon (;) to separate multiple AfterSQL statements. The SQL statement is entered in a resizable edit box. Execution occurs immediately after the last row is processed, before the data source connection is terminated. The **Before** and **After** pages look alike.

Reading Data from OLE DB

The following sections describe the differences when you use generated queries or user-defined queries to read data from a data source into DataStage.

The column definitions for reference links must contain a key field. You use key fields to join primary and reference inputs to a Transformer stage. MS OLEDB key reads the data by using a WHERE clause in the SQL SELECT statement.

Using Generated Queries

By default, DataStage extracts data from a data source using an SQL SELECT statement that it constructs. The SQL statement is automatically constructed using the table and column definitions that you entered on the **Output** page.

When you select **Yes** in **SQL generation**, data is extracted from a data source using an SQL SELECT statement constructed by DataStage. SQL SELECT statements have the following syntax:

SELECT clause FROM clause
[WHERE clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause];

When you specify the data sources to use and the columns to be output from the MS OLEDB stage, the SQL SELECT statement is automatically constructed and can be viewed by clicking the **SQL** tab on the **Output** page.

For example, if you extract the Name, Address, and Phone columns from a table called Table1, the SQL statement displayed of the **SQL** page is:

SELECT Name, Address, Phone FROM Table1;

The SELECT and FROM clauses are the minimum required and are automatically generated by DataStage. If you want to use the following additional SQL SELECT clauses, you must enter them on the **Selection** page of the **Output** page:

SELECT clause	Specifies the columns to select from the database.
FROM clause	Specifies the tables containing the selected columns.
WHERE clause	Specifies the criteria that rows must meet to be selected.
GROUP BY clause	Groups rows to summarize results.
HAVING clause	Specifies the criteria that grouped rows must meet to be selected.
ORDER BY clause	Sorts selected rows.

Using User-Defined Queries

Instead of using the SQL statement constructed by DataStage, you can enter your own SQL statement for each MS OLEDB output link. To enter an SQL statement:

1. Set **SQL generation** to **No** on the **General** page of the **Output** page. The **User-defined** page on the **SQL** page is enabled. It looks like the **User-defined** page for the input link.
2. You can edit the statements or drag and drop the selected columns into your user-defined SQL statement. You must ensure that the table definitions for the output link are correct and represent the columns that are

expected. The result set generated from this statement returns at least one row. If more than one result set is produced, only the first set is used.

3. Click **OK** to close this dialog box. Changes are saved when you save your job design.

Restrictions for using user-defined SQL queries are as follows:

- If you use multiple SQL SELECT statements to read the data, only the last statement returns the result.
- Nested SQL statements are not supported.
- If more than one result set is produced, only the first set is used.
- Rowsets resulting from the execution of BeforeSQL and AfterSQL statements are not processed.
- For reference output links, the SELECT statement should have parameter markers (?) specified in the column definitions and the "Where clause" property. The parameter markers must be in the same order as the associated key columns listed in the stage properties.

OLE DB Server Data Type Support

The following table documents the support for OLE DB character, numeric, and date data types. It summarizes the data types for DataStage SQL type definitions, and their OLE DB SQL server data types:

DataStage SQL Data Type	OLE DB Data Type
SQL_CHAR	DBTYPE_WSTR (Unicode string)
SQL_VARCHAR	DBTYPE_WSTR
SQL_STRING	DBTYPE_WSTR
SQL_LONGVARCHAR	DBTYPE_WSTR
SQL_BINARY	DBTYPE_BYTES (array of bytes)
SQL_VARBINARY	DBTYPE_BYTES
SQL_LONGVARBINARY	DBTYPE_BYTES
SQL_NUMERIC	DBTYPE_R8 (double)
SQL_DECIMAL	DBTYPE_R8

DataStage SQL Data Type	OLE DB Data Type
SQL_FLOAT	DBTYPE_R8
SQL_REAL	DBTYPE_R8
SQL_DOUBLE	DBTYPE_R8
SQL_INTEGER	DBTYPE_I4 (long)
SQL_SMALLINT	DBTYPE_I4
SQL_BIGINT	DBTYPE_I4
SQL_TINYINT	DBTYPE_I4
SQL_BIT	DBTYPE_I4
SQL_DATE	DBTYPE_DBDATE
SQL_TIME	DBTYPE_DBTIME
SQL_TIMESTAMP	DBTYPE_DBTIMESTAMP

Stored Procedure Support

You can call stored procedures from the server MS OLEDB. The following rules apply:

- Specify input parameters as literal values. Passing row values as parameter values is not supported.
- Output parameters are not supported.
- You can call stored procedures as part of the BeforeSQL and AfterSQL statements. Any result sets generated by the procedure are discarded.
- You can also call stored procedures as part of the user-defined SQL statement for all links. The stored procedure must generate a row result set that matches the stage output column definitions. Only one row result set is processed, and any additional result sets are discarded. The input link parameter count should correspond to the input action. For examples, see "Using User-Defined SQL Statements" on page 20.

CREATE CUBE Statement

The CREATE CUBE statement defines the structure of a new local cube. This statement shares much of the syntax for SQL-92 and the CREATE TABLE statement, but has added syntax for cubes. Use the INSERT INTO statement to populate the cube. For further details on the INSERT INTO statement, see "INSERT INTO Statement" on page 32.

The following sections document the syntax in Backus Naur Form (BNF) notation. For more information about cubes, see "Pivot Table Service" in the online Microsoft SQL Server for OLAP Services documentation.

CREATE CUBE Syntax in BNF Notation

The syntax for the CREATE CUBE statement uses BNF notation. BNF is a notation format using a series of symbols and production rules that successively break down statements into their components.

<create-cube-statement > ::= CREATE CUBE <cube name> <open paren>
DIMENSION <dimension name> [TYPE TIME],
<hierarchy def> [<hierarchy def>...]
[{, DIMENSION <dimension name> [TYPE TIME],
<hierarchy def> [<hierarchy def>...]}...] ,
MEASURE <measure name> <measure function def> [<measure format def>]
[<measure type def>]

[{, MEASURE <measure name> <measure function def> [<measure format def>]
[<measure type def>] }...]
[,COMMAND <expression>]
[,COMMAND <expression>...]
<close paren>
.<dimension name> ::= <legal name>
<hierarchy def> ::= [HIERARCHY <hierarchy name>,] <level def> [,<level def>...]
<level def> ::= LEVEL <level name> [TYPE <level type>] [<level format def>]
[<level options def>]
<level type> ::= ALL | YEAR | QUARTER | MONTH | WEEK | DAY | DAYOFWEEK
|DATE | HOUR | MINUTE | SECOND
<level format def> ::= FORMAT_NAME <expression> [FORMAT_KEY <expression>]
<level options def> ::= OPTIONS (<option_list>)
<option_list> :: = <option> [,<option_list>]
<option> ::= UNIQUE | SORTBYNAME | SORTBYKEY
<measure function def> ::= FUNCTION <function name>
<measure format def> ::= FORMAT <expression>
<function name> ::= SUM | MIN | MAX | COUNT
<measure type def> ::= TYPE <supported OLEDB numeric types>
<supported OLEDB numeric types> :: = DBTYPE_I1 | DBTYPE_I2 | DBTYPE_I4
| DBTYPE_I8 | DBTYPE_UI1 | DBTYPE_UI2 | DBTYPE_UI4 | DBTYPE_UI8 |
DBTYPE_R4 | DBTYPE_R8 | DBTYPE_CY | DBTYPE_DECIMAL | DBTYPE_NUMERIC
| DBTYPE_DATE

DIMENSION Clause. The name given to a TYPE ALL level applies the specified name to the ALL member rather than the ALL level. The ALL level always has the name All. For example, the clause LEVEL [All Customers] TYPE ALL creates a level named (All) containing a single member named [All Customers]. There is no [All Customers] level.

COMMAND Clause. If the <expression> value has spaces, use brackets to surround the whole expression. We do not recommend using quotation marks because the body of the command can include quotation marks. (OLAP Services supports nested brackets but not nested quotation marks.)

Example

```
CREATE CUBE Sales
(
DIMENSION Time TYPE TIME,
HIERARCHY [Fiscal],
LEVEL [Fiscal Year] TYPE YEAR,
LEVEL [Fiscal Qtr] TYPE QUARTER,
LEVEL [Fiscal Month] TYPE MONTH OPTIONS (SORTBYKEY, UNIQUE),
HIERARCHY [Calendar],
LEVEL [Calendar Year] TYPE YEAR,
```

```
LEVEL [Calendar Month] TYPE MONTH,
DIMENSION Products,
LEVEL [All Products] TYPE ALL,
LEVEL Category,
LEVEL [Sub Category],
LEVEL [Product Name],
DIMENSION Geography,
LEVEL [Whole World] TYPE ALL,
LEVEL Region,
LEVEL Country,
LEVEL City,
MEASURE [Sales]
FUNCTION SUM
FORMAT 'Currency',
MEASURE [Units Sold]
FUNCTION SUM
TYPE DBTYPE_UI4
)
```

INSERT INTO Statement

The INSERT INTO statement is similar to the SQL-92 syntax for creating and populating tables. It populates a local cube with dimension members. If the local cube is in multidimensioned (MOLAP) storage mode, the INSERT INTO statement also populates the local cube with data. The INSERT INTO statement is used after a CREATE CUBE statement to create a local cube.

Backus Naur Form (BNF) Notation

The following syntax in BNF notation documents the INSERT INTO statement.

```
<insert-into-statement> ::= INSERT INTO <target-clause>
[<options-clause>] [<bind-clause>] <source-clause>
<target-clause> ::= <cube-name> <open-paren> <target-element-list>
<close-paren>
<target-element-list> ::= <target-element>[, <target-element-list>]
<target-element> ::= [<dim-name>.[<hierarchy-name>.]]<level-name>
| <time-dim-name>
| [Measures.]<measure-name>
| SKIPONECOLUMN
<level-name> ::= <simple-level-name> | <simple-level-time>.NAME
|<simple-level-time>.KEY
<time-dim-name> ::= <dim-name-type-time> | <dim-name-type-time>.NAME
| <dim-name-type-time>.KEY
```

<options-clause> ::= OPTIONS <options-list>
<options-list> ::= <option>[, <options-list>]
<option> ::= <defer-options> | < analysis-options>
<defer-options> ::= DEFER_DATA | ATTEMPT_DEFER
<analysis-options> ::= PASSTHROUGH | ATTEMPT_ANALYSIS
<bind-clause> ::= BIND (<bind-list>)
<bind-list> ::= <simple-column-name>[,<simple-column-name>]
<simple-column-name> ::= <identifier>
<source-clause> ::= SELECT <columns-list>
FROM <tables-list>
[WHERE <where-clause>]
| DIRECTLYFROMCACHEDROWSET <hex-number>
<columns-list> ::= <column-expression> [, < columns-list>]
<column-expression> ::= <column-expression-name>
<column-expression-name> ::= <column-name> [AS <alias-name>]
| <alias name> <column-name>
<column-name> ::= <table-name>.<column-name>
| <column-function> | <ODBC scalar function> | <braced-expression>
<column function> ::= <identifier>(…)
<ODBC scalar function> ::= {FN<column-function>}
<braced-expression> ::= (…)
<tables list> ::= <table-expression> [, <table-list>]
<table-expression> ::= <table-name> [[AS] <table-alias>]
<table-alias> ::= <identifier>
<table-name> ::= <identifier>
<where clause> ::= <where-condition> [AND <where-clause>]
<where condition> ::= <join-constraint> | <application constraint>
<join-constraint> ::= <column-name> = <column-name>
| <open-paren><column-name> = <column-name><close-paren>
<application-constraint> ::= (…)

| NOT (…)
| (…) OR (…)
<identifier> ::= <letter>{<letter> | <digit> | <underline> | <dollar> | <sharp>}…

Names of Elements. These are level and measure names, sometimes qualified with dimension name or the Measures keyword to avoid ambiguity. The Measures keyword is case-sensitive in binary comparisons. If you use binary comparison or are unsure of your comparison method, use Measures as shown with initial uppercase.

Each level and each measure in a cube is derived from a column in the SELECT clause.

The Columns Specified in the Associated SELECT Clause. These are bound to the elements of the INSERT INTO statement in the order specified and in a one-to-one relationship.

Each level can be derived from two columns, with one used as a name column and the other used as a key column. Both columns must be in the same table. If there are two columns associated with a level, use the .NAME or .KEY suffix in the INSERT INTO statement after the level name.

If a column specified in the SELECT clause does not have a related element in the INSERT INTO statement, use the SKIPONECOLUMN keyword as a placeholder for the unused column. You can use SKIPONECOLUMN more than once.

Dimension of TYPE TIME. Specify by the name of the dimension. The dimension name correlates the entire dimension with a single column in the source table that contains data with a date or time data type. The TYPE <level type> levels, identified for the time dimension in the CREATE CUBE statement cause the time information to be extracted from the source column specified in the SELECT clause. See “Example 4” on page 36.

The WHERE Clause. This clause can have both application and join constraints. The parser parses only join constraints, using the join constraint to find a path from all tables to the fact table and the dimension tables. The application constraint is used only to specify constraints on a fact table and is passed through unmodified.

Expressions Between Parentheses. These expressions are treated as application constraints. For example, if the expression Sales.Product_ID = Products.Product_ID AND Sales.Customer_ID = Customers. Customer_ID is enclosed

in parentheses, it is treated as an application constraint and is not used as a join constraint. Use parentheses only around application constraints in your application, for example, (Product.Price < 100 AND Product.Category = 1).

BIND Clause. This is used to bind level and measure names specified with column names used to create rowsets.

AS <alias-name> Syntax. This is not supported for local cubes in relational (ROLAP) storage mode.

Example 1

```
INSERT INTO MyCube (Year, Month.Name, Month.Key, [Product Group],
[Product Name], Country, Sales, Cost)
OPTIONS DEFER_DATA
SELECT MyTable.Year, MyTable.Month, MONTH(MyTable.Month),
MyTable.ProdGroup, MyTable.ProdName, MyTable.Country, MyTable.Sales,
MyTable.Cost
FROM MyTable
WHERE MyTable.SalesRep = "Amir" and MyTable.CustomerGroup = "Industry"
```

Example 2

```
INSERT INTO MyCube (Year, Month, [Product Group], [Product Name], Country,
Sales, Cost)
OPTIONS PASSTHROUGH SELECT MyTable.Year, MyTable.Month,
MyTable.ProdGroup, MyTable.ProdName, MyTable.Country, MyTable.Sales,
MyTable.Cost
FROM MyTable
WHERE MyTable.SalesRep = "Amir" and MyTable.CustomerGroup = "Industry"
```

Note: The PASSTHROUGH option specifies that the SELECT clause that follows it is to be passed directly to the database engine with no parsing by Pivot Table Service.

Example 3

```
INSERT INTO MyCube (Year, Month, [Product Group], [Product Name], Country,
Sales, Cost)
DIRECTLYFROMCACHEDROWSET 0x00001284
```

Note: The DIRECTLYFROMCACHEDROWSET keyword directs data to be read from the address in memory that is identified immediately after the keyword. You must specify the correct address in memory in your application. At run time, the number is assumed to be the in-process address of an IUnknown pointer to an OLE DB rowset.

Example 4

```
CREATE CUBE MyCube (
DIMENSION TimeDim TYPE TIME,
LEVEL MyYear TYPE YEAR,
LEVEL MyQtr TYPE QUARTER,
LEVEL MyMonth TYPE MONTH,
DIMENSION Products,
LEVEL [Product Group],
LEVEL [Product Name],
DIMENSION Geography,
LEVEL State,
LEVEL City,
MEASURE [Sales]
FUNCTION SUM
FORMAT 'Currency',
MEASURE [Units Sold]
FUNCTION SUM
)
INSERT INTO MyCube (TimeDim, [Product Group], [Product Name], State, City,
Sales, [Units Sold])
OPTIONS DEFER_DATA
SELECT MyTable.TransDate, MyTable.ProdGroup, MyTable.ProdName,
MyTable.State, MyTable.City, MyTable.Sales, MyTable.UnitsSold
FROM MyTable
WHERE MyTable.SalesRep = "Jacobsen" and MyTable.CustomerGroup = "Industry"
```

Passthrough OPTION

The PASSTHROUGH option provides advanced query processing. It causes the SELECT clause to be passed directly to the source database without modification by Pivot Table Service. If PASSTHROUGH is not specified, Pivot Table Service

parses the query and formulates a set of queries equivalent to the original. These queries are optimized for the source database and index structures. This set of queries is often more efficient than the specified query.

The DEFER_DATA option causes the query to be parsed locally. It is executed only when necessary to retrieve data to satisfy a user request. DEFER_DATA specifies that a local cube is defined in the ROLAP storage mode.

The ATTEMPT_DEFER option causes Pivot Table Service to parse the query and defer data loading if successful. If the query cannot be parsed, it processes the specified query immediately as if PASSTHROUGH were specified.

The ATTEMPT_ANALYSIS option causes Pivot Table Service to parse the query and formulate an optimized set of queries (process in the MOLAP mode). If the query cannot be parsed, it processes the specified query immediately as if PASSTHROUGH were specified.

Passthrough Compatibility

The following table summarizes the options for the INSERT INTO statement for the MOLAP and ROLAP storage modes. PT indicates PASSTHROUGH functionality.

Option	Parse	Neither PASSTHROUGH Nor ATTEMPT_ ANALYSIS	PASS-THROUGH	ATTEMPT_ ANALYSIS
Neither DEFER_DATA nor ATTEMPT_ DEFER	Succeeded	MOLAP	MOLAP (PT)	MOLAP
	Failed	Error	N/A	MOLAP (PT)
DEFER_DATA	Succeeded	ROLAP	Error	ROLAP
	Failed	Error	N/A	Error
ATTEMPT_ DEFER	Succeeded	ROLAP	MOLAP (PT)	ROLAP
	Failed	MOLAP (PT)		